Reorganize the equations

💡 the review paper is really nice, but most of the equations are not relevant to my project 🤪 took some time to realize it

💡 TLDR: for each layer in iBNN, add `sum_log_abs_det_jacobian` of the NF part to the original KL divergence, keep everything else the same

Motivation for approximation and derivation of ELBO

In the first place, we have a **dataset** $\mathcal{D}$, and a deterministic **base neural network** with parameters θ.

With these, we want to infer from the data a set of latent variables $\mathbf{z}_1, ..., \mathbf{z}_l$ that control the stochastic part in each layer of the the base model.

That is, we want to know the posterior distribution of the latent variables $p(\mathbf{z}_{1:l}|\mathcal{D};\theta)$, but the true model is so complicated that we have to find something simpler, i.e., $q(\mathbf{z}_{1:l};\theta)$, to approximate it.

To this end, we need to minimize the KL divergence between the true posterior $p(\mathbf{z}_{1:l}|\mathcal{D};\theta)$ and its approximation $q(\mathbf{z}_{1:l};\theta)$

$$
\begin{aligned}
KL[q(\mathbf{z}_{1:l};\theta)||p(\mathbf{z}_{1:l}|\mathcal{D};\theta)] &= \mathbb{E}_q[\ln q(\mathbf{z}_{1:l})] - \mathbb{E}_q[\ln p(\mathbf{z}_{1:l}|\mathcal{D})] \quad (\text{omit } \theta \text{ for simplicity}) \\
&= \mathbb{E}_q[\ln q(\mathbf{z}_{1:l})] - \mathbb{E}_q[\ln \frac{p(\mathcal{D}|\mathbf{z}_{1:l})p(\mathbf{z}_{1:l})}{p(\mathcal{D})}] \quad (\text{Bayes's rule}) \\
&= \mathbb{E}_q[\ln q(\mathbf{z}_{1:l})] - \mathbb{E}_q[\ln p(\mathcal{D}|\mathbf{z}_{1:l})] - \mathbb{E}_q[\ln p(\mathbf{z}_{1:l})] + \mathbb{E}_q[\ln p(\mathcal{D})] \\
&= KL[q(\mathbf{z}_{1:l})||p(\mathbf{z}_{1:l})] - \mathbb{E}_q[\ln p(\mathcal{D}|\mathbf{z}_{1:l})] + \mathbb{E}_q[\ln p(\mathcal{D})]
\end{aligned}
$$

If we rearrange the terms

$$
\mathbb{E}_q[\ln p(\mathcal{D})] = \underbrace{\mathbb{E}_q[\ln p(\mathcal{D}|\mathbf{z}_{1:l})] - KL[q(\mathbf{z}_{1:l})||p(\mathbf{z}_{1:l})]}_{ELBO} + KL[q(\mathbf{z}_{1:l};\theta)||p(\mathbf{z}_{1:l}|\mathcal{D};\theta)]
$$

Since KL divergence is always larger than 0 and the probability of dataset is constant, maximizing ELBO is equivalent to minimizing the KL term

If we further assume that $\mathbf{z}_1, ..., \mathbf{z}_l$ are independent, $q(\mathbf{z}_{1:l})$ could be factorized into a product $q(\mathbf{z}_{1:l}) = \prod_{i=1}^{l} p(\mathbf{z}_i)$

and we can decompose the second term in $ELBO$ into a sum of $\ln p(\mathbf{z}_i)$

$$ELBO = \mathbb{E}_q[\ln p(\mathcal{D}|\mathbf{z}_{1:l})] - \beta \sum_{i=1}^{l} KL[q(\mathbf{z}_i)||p(\mathbf{z}_i)]$$

How to compose the approximation?

extend iBNN workflow by modifying the forward propagation

$$\begin{aligned}
\mathbf{f}_l(\mathbf{x}) &= \sigma_l\left(\mathbf{U}_\ell\left(\mathbf{z}_l \circ \mathbf{f}_{l-1}\right) + \mathbf{b}_l\right), \quad l \geq 1 \\
\mathbf{f}_0 &= \mathbf{x} \\
\mathbf{z}_l &= NF_l(\mathbf{z}) \\
\mathbf{z} &\sim p(\mathbf{z})
\end{aligned}$$

the difference is: put the base random variable $\mathbf{z}$ (Gaussian) through a (series of) transformation $NF_l(\cdot)$ to get a new latent variable $\mathbf{z}_l$

$$\begin{aligned}
\mathbf{z}_l &= f_l^k(f_l^{k-1}(\ldots f_l^1(\mathbf{z}))) \\
&= f_l^k \circ f_l^{k-1} \circ \ldots \circ f_l^1(\mathbf{z})
\end{aligned}$$

and the corresponding probability density will be (the product of `det_jacobian` for the random variable transformed by previous f's)

$$\begin{aligned}
q(\mathbf{z}_l) &= p(\mathbf{z})|\det J_{NF_l}(\mathbf{z})|^{-1} \\
&= p(\mathbf{z})\left|\prod_{i=1}^{k} \det J_{f_l^i}(f_l^{i-1} \circ \ldots \circ f_l^1(\mathbf{z}))\right|^{-1} \\
&= p(\mathbf{z})\prod_{i=1}^{k}\left|\det J_{f_l^i}(f_l^{i-1} \circ \ldots \circ f_l^1(\mathbf{z}))\right|^{-1}
\end{aligned}$$

Intuitively, as the base random variable goes through the series of transformations, the input to a layer of transform will be a composed random variable that has been affected by all previous layers, so it also make sense to calculate the Jacobian with respect to the immediate input of that transform.

The product of Jacobian comes from recursively applying the rule for derivative of composite functions $f(g(x))' = f'(g(x))g'(x)$

How to calculate ELBO in theory?

$$ELBO = \mathbb{E}_q[\ln p(\mathcal{D}|\mathbf{z}_{1:l})] - \beta \sum_{i=1}^{l} KL[q(\mathbf{z}_i)||p(\mathbf{z}_i)]$$

The first term is the data likelihood, and it's obtained with the prediction result (should be the same as iBNN, Categorical loss)

The second term is the KL divergence, which forces the approximation to keep close to prior and thus controlling the model complexity.

Let's take the KL of just one layer for simplicity

$$\begin{aligned} KL[q(\mathbf{z}_i)||p(\mathbf{z}_i)] &= \mathbb{E}_q[\ln q(\mathbf{z}_i)] - \mathbb{E}_q[\ln p(\mathbf{z}_i)] \\ &= \mathbb{E}_q[\ln p(\mathbf{z})] + \sum_{i=1}^{k} \mathbb{E}_q \ln \left| \det J_{f_l^i}(f_l^{i-1} \circ ... \circ f_l^1(\mathbf{z})) \right|^{-1} - \mathbb{E}_q[\ln p(\mathbf{z}_i)] \\ &= KL[p(\mathbf{z})||p(\mathbf{z}_i)] - \mathbb{E}_q \sum_{i=1}^{k} \ln \left| \det J_{f_l^i}(f_l^{i-1} \circ ... \circ f_l^1(\mathbf{z})) \right| \end{aligned}$$

the first term should be the same as iBNN, just the KL between the learned base distribution $p(\mathbf{z})$ and prior distribution $p(\mathbf{z}_i)$

the second term comes from normalizing flow (and we ONLY need this from the NF), has a long name `sum_log_abs_det_jacobian`

As for $\mathbb{E}_q$, we could average over the batch, if we use a reasonably large batch size, say 64, 128

- we could choose $p(\mathbf{z})$ to be the same as $p(\mathbf{z}_i)$, for example, they are both $N(0, 1)$, and let the inner most NF to be $y = ax + b$
- we could also allow them to be different, and the NF part could become simpler

This equation looks like the flow-based free energy bound (Equation 15 in VINF paper)

$$\begin{aligned} \mathcal{F}(\mathbf{x}) &= \mathbb{E}_{q_\phi(z|x)} \left[\log q_\phi(\mathbf{z} \mid \mathbf{x}) - \log p(\mathbf{x}, \mathbf{z}) \right] \\ &= \mathbb{E}_{q_0(z_0)} \left[\log q_K\left(\mathbf{z}_K\right) - \log p\left(\mathbf{x}, \mathbf{z}_K\right) \right] \\ &= \mathbb{E}_{q_0(z_0)} \left[\log q_0\left(\mathbf{z}_0\right) \right] - \mathbb{E}_{q_0(z_0)} \left[\log p\left(\mathbf{x}, \mathbf{z}_K\right) \right] - \mathbb{E}_{q_0(z_0)} \left[\sum_{k=1}^{K} \ln \mid 1 + \mathbf{u}_k^\top \psi_k\left(\mathbf{z}_{k-1}\right) \mid \right] \end{aligned}$$

What kind of flows are available (need the ability to evaluate density)?

(mainly from the review paper) a powerful tool is the matrix determinant lemma, somehow looks like Sherman-Morrison formula

$$\det\left(\mathbf{A} + \mathbf{V}\mathbf{W}^\top\right) = \det\left(\mathbf{I} + \mathbf{W}^\top \mathbf{A}^{-1}\mathbf{V}\right) \det \mathbf{A}$$

planar flow, before transform $\mathbf{z}$, after transform $\mathbf{z}'$

$$\begin{aligned} \mathbf{z}' &= \mathbf{z} + \mathbf{v}\sigma\left(\mathbf{w}^\top \mathbf{z} + b\right) \\ J_{f_\phi}(\mathbf{z}) &= \mathbf{I} + \sigma'\left(\mathbf{w}^\top \mathbf{z} + b\right)\mathbf{v}\mathbf{w}^\top \\ \det J_{f_\phi}(\mathbf{z}) &= 1 + \sigma'\left(\mathbf{w}^\top \mathbf{z} + b\right)\mathbf{w}^\top \mathbf{v} \end{aligned}$$

Sylvester flow

$$
\begin{aligned}
\mathbf{z}' &= \mathbf{z} + \mathbf{V}\sigma\left(\mathbf{W}^\top \mathbf{z} + \mathbf{b}\right) \\
J_{f_\phi}(\mathbf{z}) &= \mathbf{I} + \mathbf{V}\mathbf{S}(\mathbf{z})\mathbf{W}^\top \\
\det J_{f_\phi}(\mathbf{z}) &= \det\left(\mathbf{I} + \mathbf{S}(\mathbf{z})\mathbf{W}^\top \mathbf{V}\right)
\end{aligned}
$$

Radial flow

$$
\begin{aligned}
\mathbf{z}' &= \mathbf{z} + \frac{\beta}{\alpha + r(\mathbf{z})}(\mathbf{z} - \mathbf{z}_0) \qquad \text{where} \quad r(\mathbf{z}) = \|\mathbf{z} - \mathbf{z}_0\| \\
J_{f_\phi}(\mathbf{z}) &= \left(1 + \frac{\beta}{\alpha + r(\mathbf{z})}\right)\mathbf{I} - \frac{\beta}{r(\mathbf{z})(\alpha + r(\mathbf{z}))^2}(\mathbf{z} - \mathbf{z}_0)(\mathbf{z} - \mathbf{z}_0)^\top \\
\det J_{f_\phi}(\mathbf{z}) &= \left(1 + \frac{\alpha\beta}{(\alpha + r(\mathbf{z}))^2}\right)\left(1 + \frac{\beta}{\alpha + r(\mathbf{z})}\right)^{D-1}
\end{aligned}
$$

Batch norm, works like a flow

$$
\begin{aligned}
\mathrm{BN}(\mathbf{z}) &= \boldsymbol{\alpha} \odot \frac{\mathrm{z} - \hat{\boldsymbol{\mu}}}{\sqrt{\hat{\sigma}^2 + \epsilon}} + \boldsymbol{\beta}, \quad \mathrm{BN}^{-1}(\mathbf{z}') = \hat{\boldsymbol{\mu}} + \frac{\mathbf{z}' - \boldsymbol{\beta}}{\boldsymbol{\alpha}} \odot \sqrt{\hat{\sigma}^2 + \boldsymbol{\epsilon}} \\
\det J_{\mathrm{BN}}(\mathbf{z}) &= \prod_{i=1}^{D} \frac{\alpha_i}{\sqrt{\hat{\sigma}_i^2 + \epsilon_i}}
\end{aligned}
$$

How to code up the extended part?

get `sum_log_abs_det_jacobian` according to the formulas, and then subtract this term to the `.kl()` function of iBNN layers, and should be ok?

```
# https://github.com/trungtrinh44/ibnn/blob/master/models/utils.py
# class BayesianLinear(nn.Linear, BayesianLayer):
def kl(self):
    prior = D.Normal(self.prior_mean, self.prior_std)
    if self.bias is not None:
        weight_sampler, bias_sampler = self.posterior()
        return D.kl_divergence(weight_sampler, prior).sum() + D.kl_divergence(bias_sampler, prior).sum()
    weight_sampler = self.posterior()
    # sum_log_abs_det_jacobian = ???
    return D.kl_divergence(weight_sampler, prior).sum() - sum_log_abs_det_jacobian
```

Questions remains

▼ how do we know if the trained model is multi-modal? (can we generate samples and plot one or two dimensions?)

▼ what if the trained model is still unimodal (because the data don't lead to a multimodal solution) even if the model supports multimodal inference in theory?

▼ If that is the case, do we have some technique to force the model to be multimodal?

▼ A guess about why the real-Bayesian DNN could be bad (implied in the paper "How Good is the Bayes Posterior in Deep Neural Networks Really?")

In a REAL Bayesian DNN, we need to compute an integral in order to make an inference

$$p(y \mid x, \mathcal{D}) = \int p(y \mid x, w) p(w \mid \mathcal{D}) dw$$

we will average over all possible values of w

even if multiple w's are equally good, it's likely that most values are bad (especially those close to 0 or infinity)

in an ideal case, we can assign tiny probability to all bad values, and they don't affect the final result at all

but it's possible that we don't have that perfect strategy to assign probability

and thus the side effect of averaging over infinitely-many bad models, would probably be non-negligible